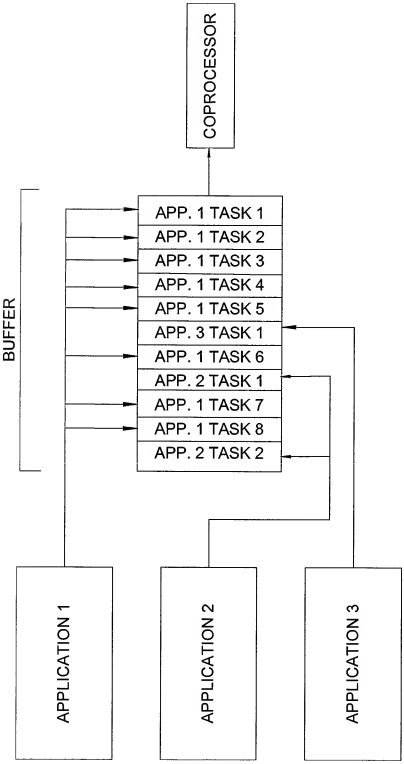


FIGURE 1
(prior art)

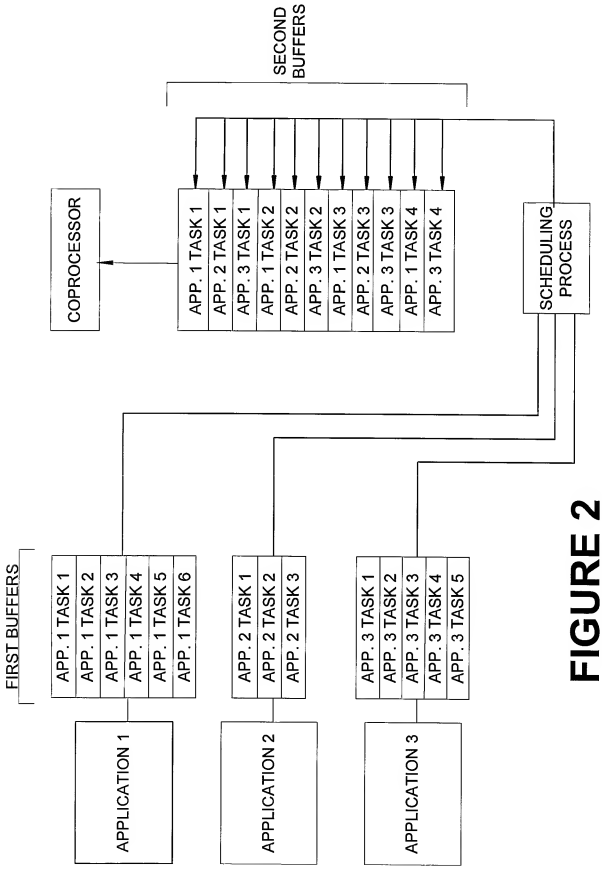

FIRST BUFFERS
APP. 1 TASK 1
APP. 1 TASK 2
APP. 1 TASK 3
APP. 1 TASK 4
APP. 1 TASK 5
APP. 1 TASK 6
APP. 2 TASK 1
APP. 2 TASK 2
APP. 2 TASK 3
APP. 3 TASK 1
APP. 3 TASK 2
APP. 3 TASK 3
APP. 3 TASK 4
APP. 3 TASK 5
APPLICATION 1
APPLICATION 2
APPLICATION 3
SCHEDULING PROCESS
SECOND BUFFERS
APP. 1 TASK 1
APP. 2 TASK 1
APP. 3 TASK 1
APP. 1 TASK 2
APP. 2 TASK 2
APP. 3 TASK 2
APP. 1 TASK 3
APP. 2 TASK 3
APP. 3 TASK 3
APP. 1 TASK 4
APP. 3 TASK 4
COPROCESSOR

FIGURE 2

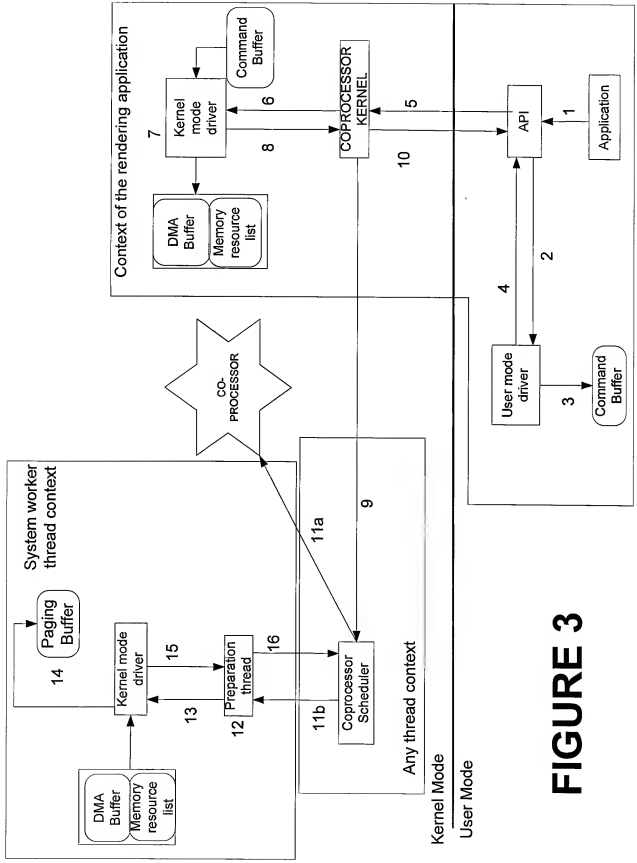
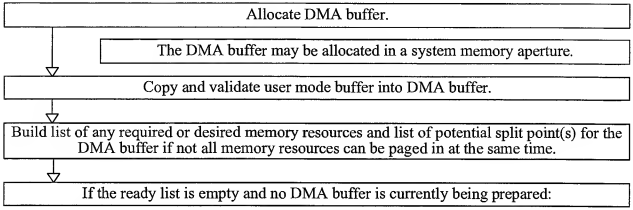

Context of the rendering application
Kernel mode driver
Command Buffer
DMA Buffer
Memory resource list
COPROCESSOR KERNEL
API
Application
User mode driver
Command Buffer
CO-PROCESSOR
System worker thread context
Paging Buffer
Kernel mode driver
Preparation thread
DMA Buffer
Memory resource list
Coprocessor Scheduler
Any thread context
Kernel Mode
User Mode
1
2
3
4
5
6
7
8
9
10
11a
11b
12
13
14
15
16

FIGURE 3

Exemplary algorithm

PROCESS A: Submit (irql passive, rendering thread context)



PROCESS B: Quantum expires (irql device, any thread context)

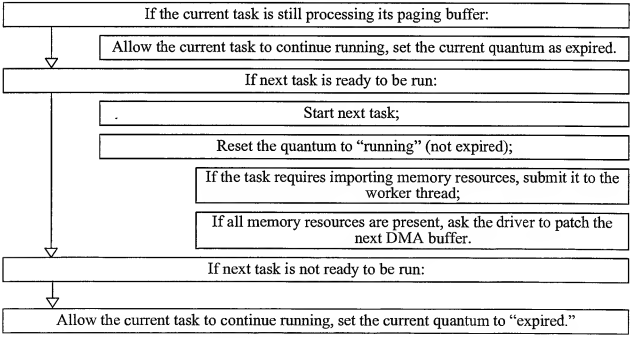


FIGURE 4(A)

Exemplary algorithm

PROCESS C: Task finishes (irql device, any thread context)

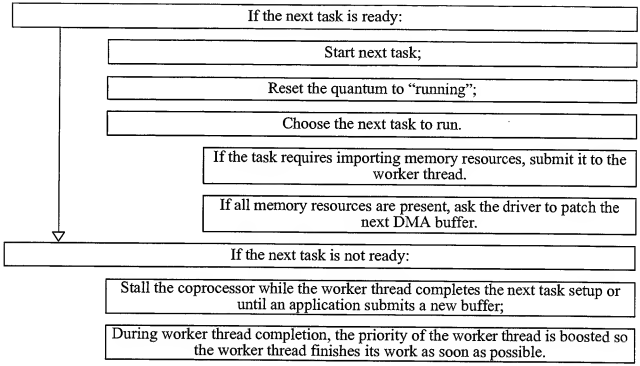
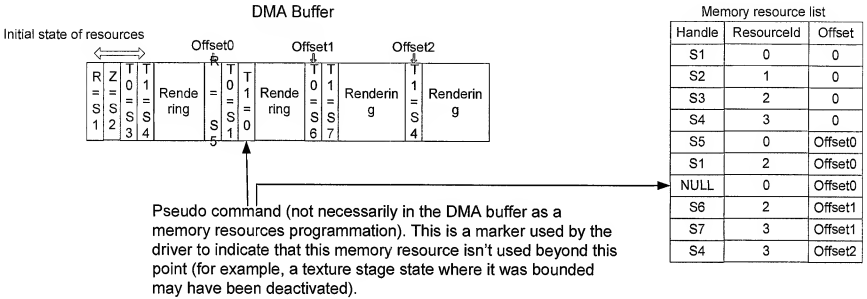
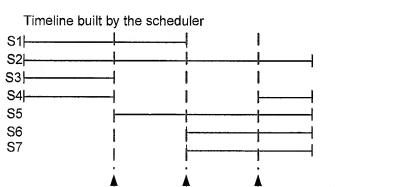


FIGURE 4(B)





Legend
R = Render target
Z = Z-buffer
T0 = Texture in stage 0
T1 = Texture in stage 1
S# = Memory resources #

FIGURE 5

Exemplary algorithm

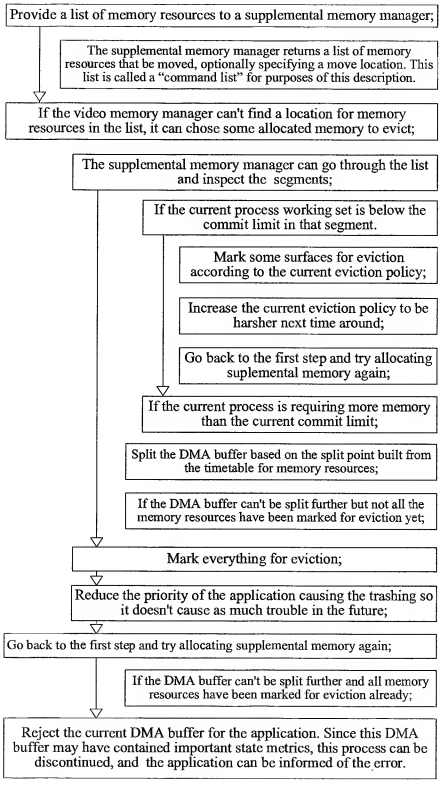


FIGURE 6

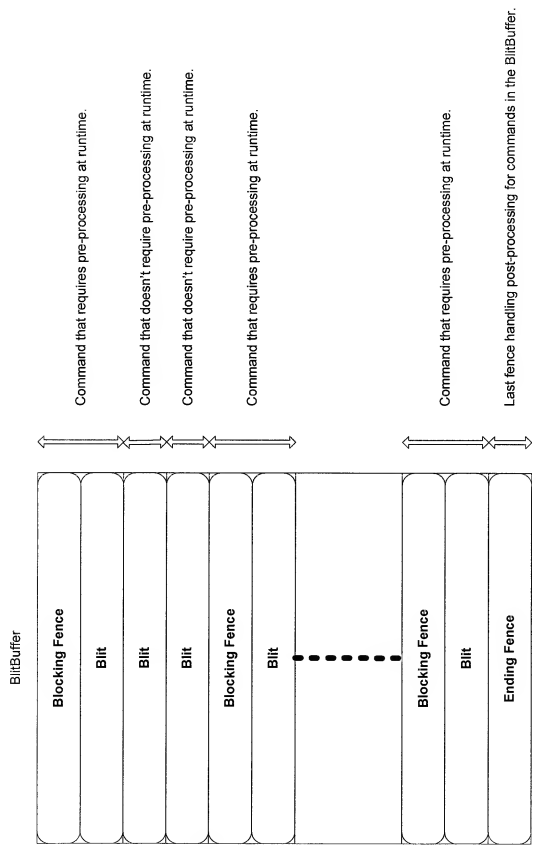

BlitBuffer
Blocking Fence
Blit
Blit
Blit
Blocking Fence
Blit
Blocking Fence
Blit
Ending Fence
Command that requires pre-processing at runtime.
Command that doesn't require pre-processing at runtime.
Command that doesn't require pre-processing at runtime.
Command that requires pre-processing at runtime.
Command that requires pre-processing at runtime.
Last fence handling post-processing for commands in the BlitBuffer.

FIGURE 7

Exemplary algorithm

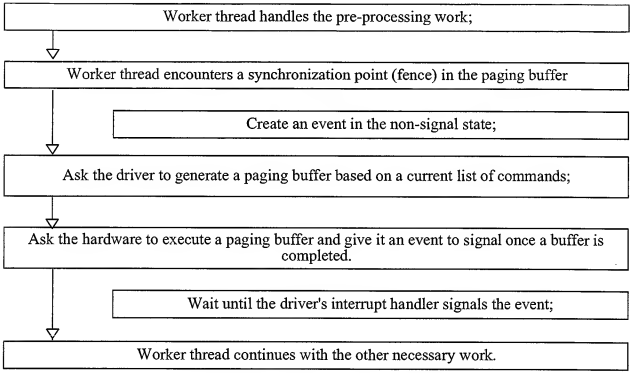


FIGURE 8

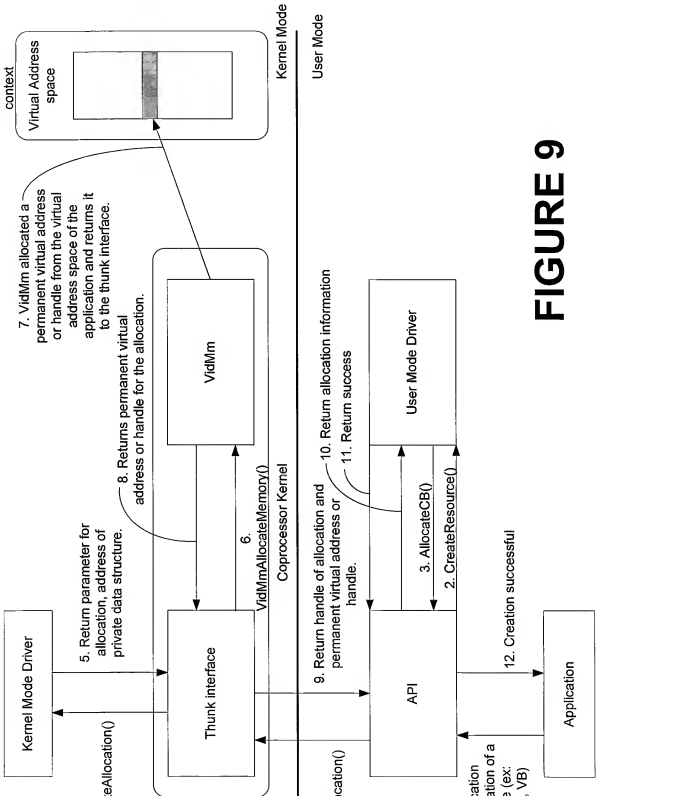

Application's context
Virtual Address space
Kernel Mode
User Mode
Kernel Mode Driver
4. CreateAllocation()
5. Return parameter for allocation, address of private data structure.
Thunk interface
VidMm
7. VidMm allocated a permanent virtual address or handle from the virtual address space of the application and returns it to the thunk interface.
8. Returns permanent virtual address or handle for the allocation.
6. VidMmAllocateMemory()
Coprocessor Kernel
4. CreateAllocation()
9. Return handle of allocation and permanent virtual address or handle.
10. Return allocation information
11. Return success
API
User Mode Driver
3. AllocateCB()
2. CreateResource()
1. Application request creation of a resource (ex: Texture, VB)
12. Creation successful
Application

FIGURE 9

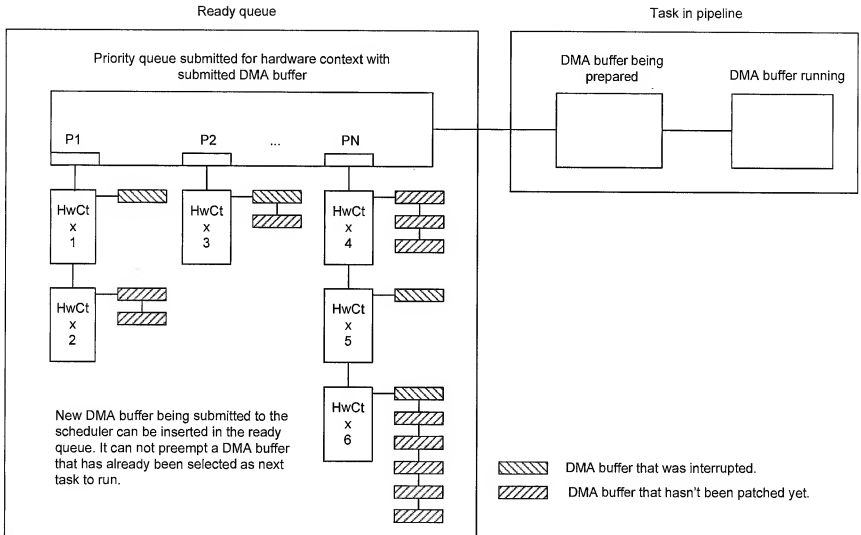

Ready queue
Priority queue submitted for hardware context with submitted DMA buffer
P1
P2
...
PN
HwCtx 1
HwCtx 3
HwCtx 4
HwCtx 2
HwCtx 5
HwCtx 6
New DMA buffer being submitted to the scheduler can be inserted in the ready queue. It can not preempt a DMA buffer that has already been selected as next task to run.
Task in pipeline
DMA buffer being prepared
DMA buffer running
DMA buffer that was interrupted.
DMA buffer that hasn't been patched yet.

FIGURE 10

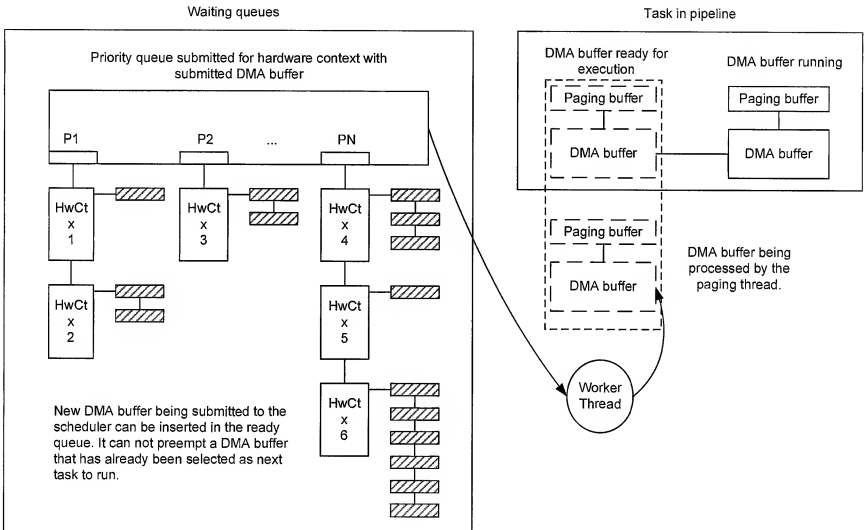

Waiting queues
Priority queue submitted for hardware context with submitted DMA buffer
P1
P2
...
PN
HwCtx 1
HwCtx 2
HwCtx 3
HwCtx 4
HwCtx 5
HwCtx 6
New DMA buffer being submitted to the scheduler can be inserted in the ready queue. It can not preempt a DMA buffer that has already been selected as next task to run.
Task in pipeline
DMA buffer ready for execution
DMA buffer running
Paging buffer
DMA buffer
Paging buffer
DMA buffer
Paging buffer
DMA buffer
DMA buffer being processed by the paging thread.
Worker Thread

FIGURE 11

PROCESS A: Submit (IRQL passive, rendering thread context)

If no DMA buffer is being prepared or is ready for execution.
    If all the memory resources for the current DMA buffer are already present in memory,
        If the coprocessor is idle, give the DMA buffer to the coprocessor.
        Else insert the DMA buffer in the ready-to-execute slot.
    If some memory resources need to be paged in, submit the DMA buffer to the paging thread.
Else, insert the DMA buffer at the end of the list for the current context.

PROCESS B: Quantum expires (IRQL device, any thread context)

If the current task is still processing its paging buffer,
    Allow the current task to continue running.
    Set the current quantum as expired.
Else, if next DMA buffer is ready to be run,
    Reset the current priority of the current context to its base priority.
    Move the current context to the end of the queue for its priority.
    Submit next DMA buffer to the coprocessor.
    Reset the quantum as being running (not expired).
    Choose the next DMA buffer to execute.
        If the DMA buffer requires paging, submit it to the paging thread.
        Else, all memory resources are already present; just insert the DMA buffer in the ready slot.
Else, the next task isn't ready to be run;
    Allow the current task to continue running.
    Set the current quantum as expired.

PROCESS C: Task finishes (IRQL device, any thread context)

If next DMA buffer is ready to be run,
    Submit next DMA buffer to the coprocessor.
    Reset the quantum as being running (not expired).
    Choose the next DMA buffer to execute.
        If the DMA buffer requires paging, submit it to the paging thread.
        Else, all memory resources are already present; just insert the DMA buffer in the ready slot.
Else, the next task isn't ready;
    If the paging thread is currently working on the next DMA buffer, boost the priority of the worker thread temporarily so it finishes its work as soon as possible.

FIGURE 12(A)

Docket No.: MSFT-3008/304862.03 App No.: 10/763,778 Filed: January 22, 2004
Title: SYSTEMS AND METHODS FOR ENHANCING PERFORMANCE OF A COPROCESSOR
Inventors: Anuj B. Gosalia and Steve Pronovost
Attorney: Vincent J. Roccia Phone: (215) 568-3100 Replacement Sheet 13 of 30

PROCESS D: Paging thread (IRQL passive, system thread)

Set current eviction policy to first policy.
Ask the memory manager to page in the resource list.
If all the resource were paged in successfully,
    Move the paging buffer and DMA buffer to the ready-to-execute slot.
    If the quantum of the current DMA buffer is expired
        Submit next DMA buffer to the coprocessor.
        Reset the quantum as being running (not expired).
        Choose the next DMA buffer to execute.
            If the DMA buffer requires paging, submit it to the paging thread.
            Else, all memory resources are already present, just insert the DMA buffer in the ready slot;
Else if the memory manager failed because the paging buffer is full
    Wait until the current DMA buffer's quantum end or finishes.
    Submit the paging buffer to the coprocessor.
    Wait until the paging buffer is done.
    Go back asking the memory manager to paged-in the remaining of the resource list.
Else if the memory manager failed because there isn't enough available resource
    If we've passed the last eviction policy
        Undo the resource move, or run the paging buffer.
        Reject the DMA buffer.
        We're done.
    Else if the current eviction policy is above application interference.
        If the DMA buffer hasn't been split yet.
            Split the DMA buffer at the closest point to the current paged-in resources.
            If no more resources are needed
                Move the paging buffer and split DMA buffer to the ready-to-execute slot.
                Move the remaining DMA buffer back to the head of the ready queue for the context.
                If the quantum of the current DMA buffer is expired,
                    Submit next DMA buffer to the coprocessor.
                    Reset the quantum as being running (not expired).
                    Choose the next DMA buffer to execute.
                        If the DMA buffer requires paging, submit it to the paging thread.
                        Else, all memory resources are already present; just insert the DMA buffer in the ready slot;.
    Ask VidMm to mark candidate for eviction using the current policy.
        If VidMm returns an error saying no memory could be marked with the current policy,
            Increase the eviction policy.
            Go back to the start of the eviction policy check.
        Else, some memory was marked.
            Go back to trying to page in the resources.

FIGURE 12(B)

Docket No.: MSFT-3006/304862.03 App. No.: 10/763,778 Filed: January 22, 2004
Title: SYSTEMS AND METHODS FOR ENHANCING PERFORMANCE OF A COPROCESSOR
Inventors: Anuj B. Gosalia et al.
Attorney: Vincent J. Rocca Phone: (215) 568-3100 Replacement Sheet 14 of 30

Supplemental memory
offset
PAGE
PAGE
System Memory
PAGE
Coprocessor Virtual address
64
11
0
BaseAddr
offset
Per-context coprocessor page table
64
11
0
BaseAddr
Reserved
c s v
Per context linear address space
Surface #1
Surface #2
DMA Buffer
COPRO-CESSOR
Coprocessor fetching from DMA buffer
Src = Surface #1
Dst = Surface #2
Blt
Valid bit
Indicates the entry refer to a valid address. (As an alternative to a valid bit, invalid entry could reference a dummy page)
System memory bit
Indicates the base address refer to a system memory location.
Snooping bit / AGP bit
Snooping bit may be PCI Express only. It indicates that the system memory can be accessed using a snooped cycle.
AGP bit may be AGP only. It indicates that the AGP cycle can be used to access the system memory.

FIGURE 13

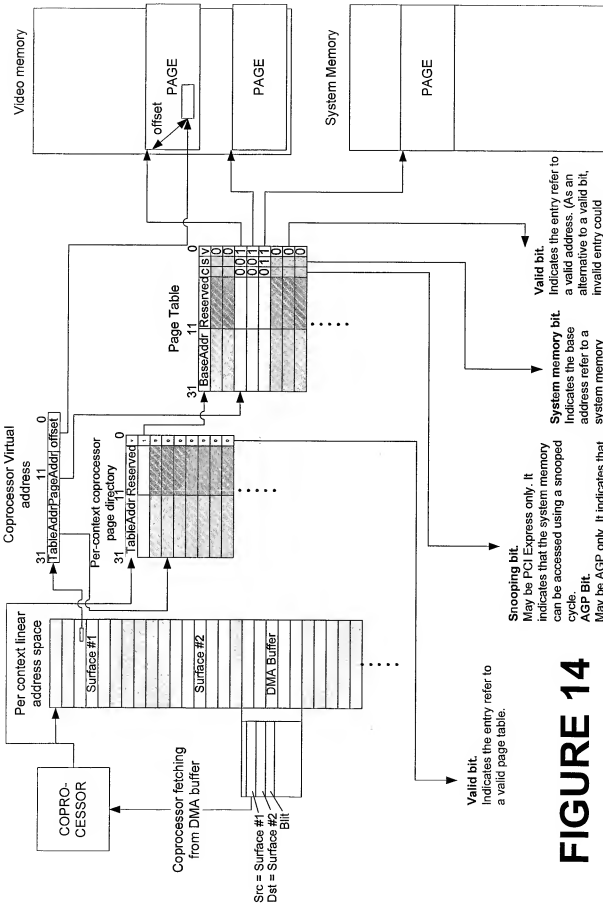

Video memory
offset
PAGE
PAGE
System Memory
PAGE
Coprocessor Virtual address
31
11
0
TableAddr
PageAddr
offset
Per-context coprocessor page directory
TableAddr
Reserved
Page Table
BaseAddr
Reserved
c
s
v
Per context linear address space
Surface #1
Surface #2
DMA Buffer
COPRO-CESSOR
Coprocessor fetching from DMA buffer
Src = Surface #1
Dst = Surface #2
Blt
Valid bit.
Indicates the entry refer to a valid page table.
Snooping bit.
May be PCI Express only. It indicates that the system memory can be accessed using a snooped cycle.
AGP Bit.
May be AGP only. It indicates that AGP cycle can be used to access the system memory.
System memory bit.
Indicates the base address refer to a system memory location.
Valid bit.
Indicates the entry refer to a valid address. (As an alternative to a valid bit, invalid entry could reference a dummy page)

FIGURE 14

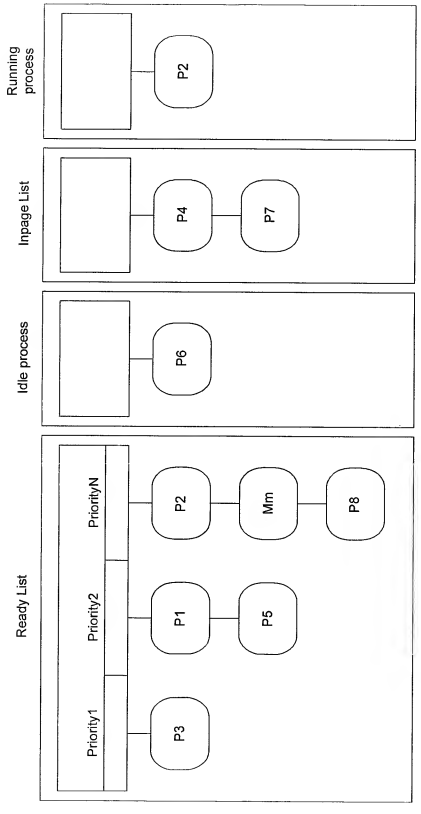
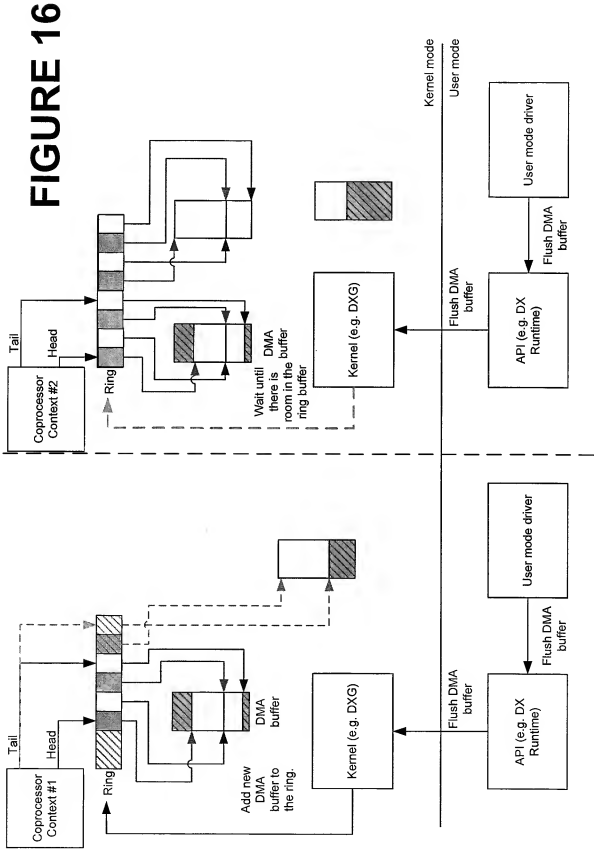

Ready List
Priority1
Priority2
PriorityN
P3
P1
P2
P5
Mm
P8
Idle process
P6
Inpage List
P4
P7
Running process
P2

FIGURE 15

FIGURE 16


Insertion in context of thread A
Insertion in context of thread B
Kernel mode
User mode
Coprocessor Context #1
Coprocessor Context #2
Tail
Head
Ring
Add new DMA buffer to the ring.
Wait until there is room in the ring buffer
DMA buffer
Kernel (e.g. DXG)
Flush DMA buffer
API (e.g. DX Runtime)
User mode driver
Flush DMA buffer

PROCESS A: Submit (IRQL passive, rendering thread context, coprocessor Context mutex held)

Acquire the VIDMM_lock.
Process the list of resources given, and update the usage information about allocations in this process.
Release the VIDMM lock.
Take the scheduler lock.
Call the driver to insert the current DMA buffer into the ring.
If the driver succeeded.
    If the context was idle.
        Insert the context back in the ready list at the tail of the queue for its current priority.
    If there is no context transfer pending and the current context is lower priority than the current context.
        Call the driver to context switch to this context.
        Signal that a context switch is pending.
Release the scheduler lock.
If the driver failed, the ring was full.
    Wait on an event that will be signaled when room becomes available.
    After the wait, go back to acquiring the scheduler lock.
If there is enough room left in the DMA buffer for another submission.
    Return to user mode with the current DMA buffer.
Acquire a new DMA buffer from the context's pool.
If DMA pool couldn't give another buffer at this time
    Wait on an event that will be signaled when a DMA buffer is inserted back into the pool.
    When the wait is over, go back to trying to get a new DMA buffer.
Return the new DMA buffer to user mode.

PROCESS B: Context switch done (IRQL device, any thread context)

Take the scheduler lock.
    If a higher priority context is now ready for execution.
        Call the driver to context switch to the highest priority context.
    Else
        Signal that no context switch is currently pending.
Release the scheduler lock

FIGURE 17(A)

PROCESS C: Quantum expires (IRQL device, any thread context)

Take the scheduler lock.
Reset the current priority of the context to its base priority.
Insert the context back at the end of the queue for its current priority.
If no context switches are currently pending.
    Ask the driver to do a context switch to the highest priority context.
Release the scheduler lock.

PROCESS A: Submit (IRQL passive, rendering thread context, coprocessor Context mutex held)

Acquire the VIDMM_lock.
Process the list of resources given, and update the usage information about allocations in this process.
Release the VIDMM lock.
Take the scheduler lock.
Call the driver to insert the current DMA buffer into the ring.
If the driver succeeded.
    If the context was idle.
        Insert the context back in the ready list at the tail of the queue for its current priority.
    If there is no context transfer pending and the current context is lower priority than the current context.
        Call the driver to context switch to this context.
        Signal that a context switch is pending.
Release the scheduler lock.
If the driver failed, the ring was full.
    Wait on an event that will be signaled when room becomes available.
    After the wait, go back to acquiring the scheduler lock.
If there is enough room left in the DMA buffer for another submission.
    Return to user mode with the current DMA buffer.
Acquire a new DMA buffer from the context's pool.
If DMA pool couldn't give another buffer at this time
    Wait on an event that will be signaled when a DMA buffer is inserted back into the pool.
    When the wait is over, go back to trying to get a new DMA buffer.
Return the new DMA buffer to user mode.

PROCESS B: Context switch done (IRQL device, any thread context)

Take the scheduler lock.
If a higher priority context is now ready for execution.
        Call the driver to context switch to the highest priority context.
Else
        Signal that no context switch is currently pending.
Release the scheduler lock

FIGURE 17(A)

PROCESS C: Quantum expires (IRQL device, any thread context)

Take the scheduler lock.
Reset the current priority of the context to its base priority.
Insert the context back at the end of the queue for its current priority.
If no context switches are currently pending.
    Ask the driver to do a context switch to the highest priority context.
Release the scheduler lock.

Docket No.: MSFT-3008/304862.03 App. No.: 10/763,778 Filed: January 22, 2004
Title: SYSTEMS AND METHODS FOR ENHANCING PERFORMANCE OF A COPROCESSOR
Inventors: Anuj B. Gosalia and Steve Pronovost
Attorney: Vincent J. Roccia Phone: (215) 568-3100 Replacement Sheet 19 of 30

PROCESS G: In page worker thread

Go through the list of contexts in the inpage queue. Pick up the highest priority one.
Ask the driver for the list of resources required to make forward progress on the context.
Take the VIDMM lock.
Find a location for each of the allocations required for forward progress.
Invalidate the virtual address or handle for the allocation getting evicted.
Ask the driver to fill a DMA buffer with the memory transfer commands necessary to bring the required allocations to their selected spots.
Release the VIDMM lock.
Submit the VidMm context as a regular coprocessor context.
If the list of contexts is empty, sleep until an item gets added.
Go back to the beginning of the loop.

PROCESS H: Periodic timer (passive level, system thread context)

Take the scheduler lock.
Increase the current priority of each context.
Release the scheduler lock.

FIGURE 17(C)

FIGURE 18

INTERRUPT
GPU
EXECUTION
A
B
C
When A completes (e.g., because its time slice is up or because of a page fault), an interrupt is raised to the coprocessor. Then A is moved to the back of the run list. Advantageously, the coprocessor can begin working on B immediately.
When a low priority thread D is started, it is appended to the run list.
In some embodiments, if D is high priority, it may inserted into the run list based on its priority (i.e., not necessarily at the end)
D

FIGURE 19

Current Run List
COPRO-CESSOR
1
2
3
4
5
New Run List
2
3
4
5
1
Currently executing context
History Length
(Must be at least twice the size of the run list)
History Base Address
(Aligned 4K)
1,Empty
2, New run list
2, Page fault
3, Page fault
Coprocessor Write Pointer
(Always refers to the address pass the last valid history data)
31
3
0
Context cookie
(aligned on 16bytes)
Reason
[31:4] Context cookie: Various embodiments may use a 32 bit value, aligned on 16 bytes, associated with a context (part of a context definition). This allows a scheduler to identify the context quickly.
[3:0] Reason: Reason for the context switch:
0: Page fault
1: Empty
2: Synchronization object
3: New run list
4: General protection fault.

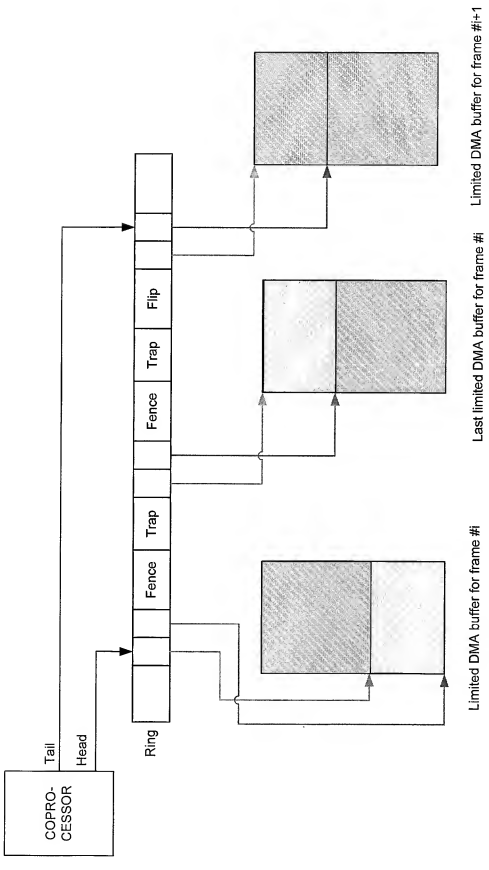

COPRO-CESSOR
Tail
Head
Ring
Fence
Trap
Fence
Trap
Flip
Limited DMA buffer for frame #i
Last limited DMA buffer for frame #i
Limited DMA buffer for frame #i+1

FIGURE 20

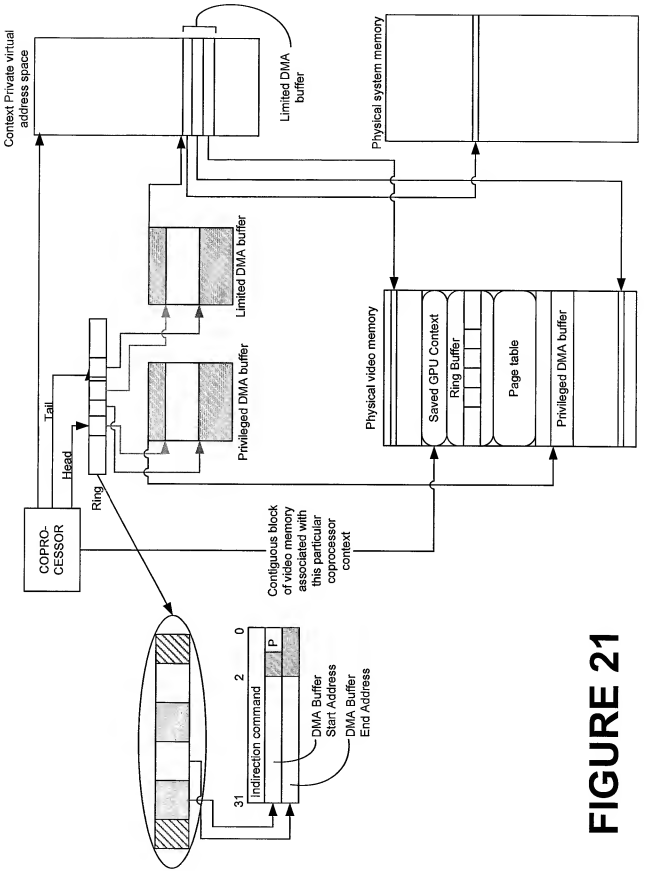

Context Private virtual address space
Limited DMA buffer
Physical system memory
Limited DMA buffer
Tail
Head
Ring
Privileged DMA buffer
Physical video memory
Saved GPU Context
Ring Buffer
Page table
Privileged DMA buffer
COPRO-CESSOR
Contiguous block of video memory associated with this particular coprocessor context
31
2
0
P
indirection command
DMA Buffer Start Address
DMA Buffer End Address

FIGURE 21

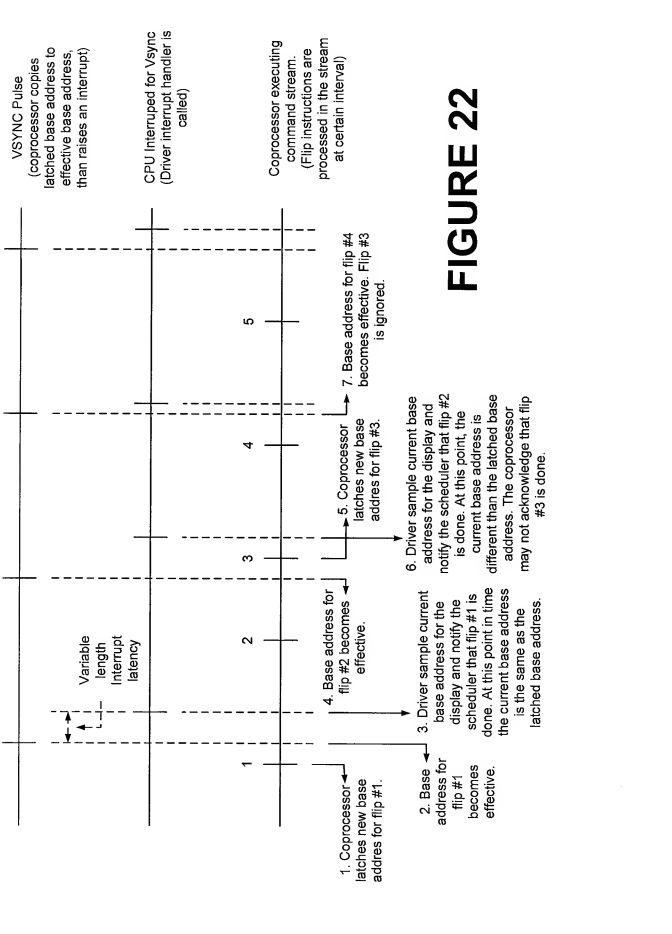

VSYNC Pulse (coprocessor copies latched base address to effective base address, than raises an interrupt)
CPU Interrupted for Vsync (Driver interrupt handler is called)
Coprocessor executing command stream. (Flip instructions are processed in the stream at certain interval)
Variable length Interrupt latency
1
2
3
4
5
1. Coprocessor latches new base addres for flip #1.
2. Base address for flip #1 becomes effective.
3. Driver sample current base address for the display and notify the scheduler that flip #1 is done. At this point in time the current base address is the same as the latched base address.
4. Base address for flip #2 becomes effective.
5. Coprocessor latches new base addres for flip #3.
6. Driver sample current base address for the display and notify the scheduler that flip #2 is done. At this point, the current base address is different than the latched base address. The coprocessor may not acknowledge that flip #3 is done.
7. Base address for flip #4 becomes effective. Flip #3 is ignored.

FIGURE 22

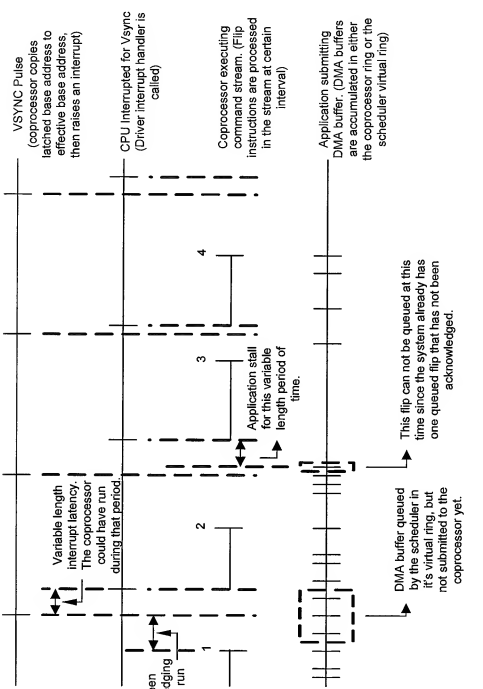

VSYNC Pulse (coprocessor copies latched base address to effective base address, then raises an interrupt)
CPU Interrupted for Vsync (Driver interrupt handler is called)
Coprocessor executing command stream. (Flip instructions are processed in the stream at certain interval)
Application submitting DMA buffer. (DMA buffers are accumulated in either the coprocessor ring or the scheduler virtual ring)
Variable length interrupt latency: The coprocessor could have run during that period.
Variable length time between reading the flip and acknowledging it. The coprocessor can not run during that period.
Application stall for this variable length period of time.
DMA buffer queued by the scheduler in it's virtual ring, but not submitted to the coprocessor yet.
This flip can not be queued at this time since the system already has one queued flip that has not been acknowledged.
1
2
3
4

FIGURE 23

Coprocessor Thread A

Pseudo code:

```
// Wait until we have exclusive access to the shared surface.
//
DxAcquireMutex(gSharedMutex);

// Set the shared surface as the render target.
//
DxSetRenderTarget(gSharedSurface);

// Render what we need in the shared surface.
//
DxDrawSomething();

// We're done with rendering, release the mutex.
//
DxReleaseMutex(gSharedMutex)
```

Coprocessor stream:

Coprocessor Thread B

Pseudo code:

```
// Wait until we have exclusive access to the shared surface.
//
DxAcquireMutex(gSharedMutex);

// Set the shared surface as a texture.
//
DxSetTexture(gSharedSurface);

// Render what we need with the shared surface.
//
DxDrawSomething();

// We're done with rendering, release the mutex.
//
DxReleaseMutex(gSharedMutex)
```

Coprocessor stream:

FIGURE 24

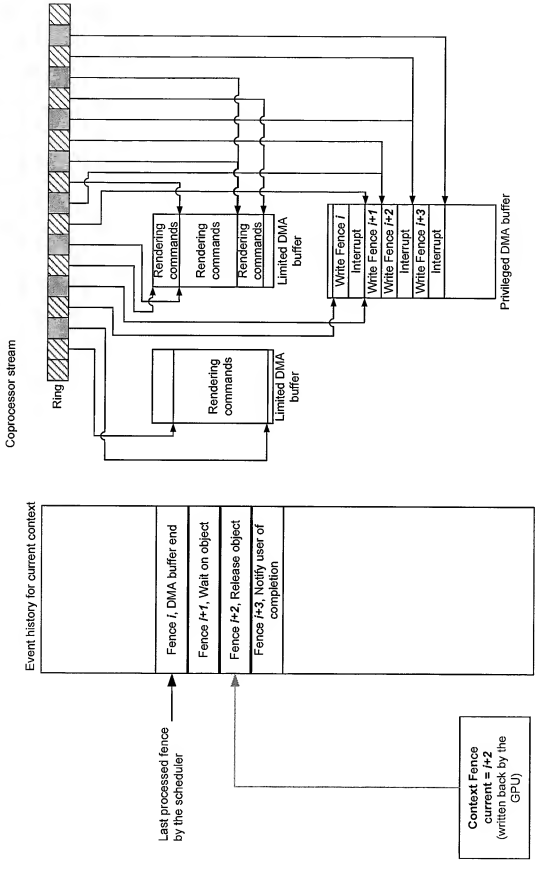

Coprocessor stream
Ring
Rendering commands
Limited DMA buffer
Write Fence i
Interrupt
Write Fence i+1
Write Fence i+2
Interrupt
Write Fence i+3
Interrupt
Privileged DMA buffer
Event history for current context
Fence i, DMA buffer end
Fence i+1, Wait on object
Fence i+2, Release object
Fence i+3, Notify user of completion
Last processed fence by the scheduler
Context Fence current = i+2 (written back by the GPU)

FIGURE 25

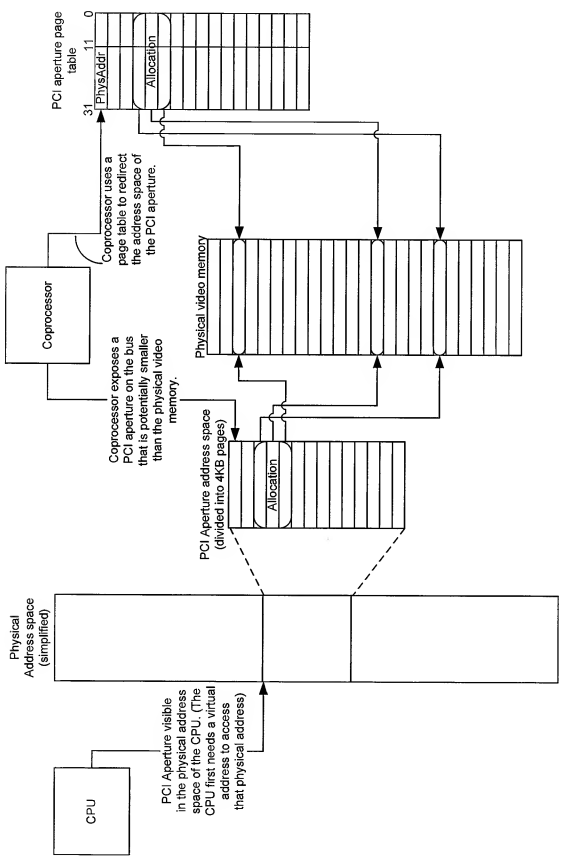

CPU
PCI Aperture visible in the physical address space of the CPU. (The CPU first needs a virtual address to access that physical address)
Physical Address space (simplified)
PCI Aperture address space (divided into 4KB pages)
Allocation
Coprocessor exposes a PCI aperture on the bus that is potentially smaller than the physical video memory.
Coprocessor
Physical video memory
Coprocessor uses a page table to redirect the address space of the PCI aperture.
PCI aperture page table
31
11
0
PhysAddr
Allocation

FIGURE 26